In blu sono indicate le risposte agli esercizi. Naturalmente per gli esercizi 1, 2, 3, in cui è richiesta la scrittura di codice, sono possibili differenti soluzioni.

In rosso sono riportati alcuni commenti relativi all'esercizio e alla soluzione.

Cognome.....

Nome.....

Matricola.....

Programmazione

Prova scritta del 4 febbraio 2014

TEMPO DISPONIBILE: 2 ore

Negli esercizi proposti si utilizzano le seguenti classi:

- Classe astratta `Number`: ogni oggetto della classe rappresenta un numero. La classe possiede un costruttore privo di argomenti. Nelle librerie standard alcune classi involucro (ad esempio `Integer`, `Long`, `Float` e `Double`) sono definite estendendo `Number`.
- Classe `InsiemeNumeri`: ogni oggetto della classe rappresenta un insieme di oggetti `Number`. Tra i metodi forniti dalla classe vi sono:
  - `public Integer maxInteger()`
    Restituisce il riferimento all'oggetto di tipo `Integer` che rappresenta il valore più grande tra tutti gli oggetti `Integer` presenti nell'insieme. Se l'insieme non contiene nessun oggetto `Integer` il metodo restituisce `null`.
  - `public Integer minInteger()`
    Analogo al precedente, per determinare il minimo `Integer` nell'insieme.
  - `public Integer diffMaxMinInteger()`
    Restituisce la differenza tra il massimo e il minimo valore `Integer` presenti nell'insieme.

1. Scrivete l'implementazione del metodo `diffMaxMinInteger`, *senza conoscere* l'implementazione di `InsiemeNumeri`, ma utilizzando gli altri metodi forniti dalla classe. In questa versione supponete che l'insieme contenga sempre almeno un valore di tipo `Integer`.

```
public Integer diffMaxMinInteger() {
  return this.maxInteger() - this.minInteger();
}
```

I riferimenti this utilizzati per richiamare i metodi possono essere omessi.

2. Riscrivete quanto richiesto per l'esercizio precedente, in modo che il metodo `diffMaxMinInteger` sollevi una eccezione di tipo `RuntimeException` se l'insieme non contiene alcun valore di tipo `Integer` (la classe `RuntimeException` fornisce un costruttore che riceve come argomento una stringa).

```
public Integer diffMaxMinInteger() {
  Integer min = this.minInteger();
  if (min == null)
    throw new RuntimeException("Non ci sono Integer!");
  else return this.maxInteger() - min;
}
```

Attenzione alla differenza tra "sollevare" e "intercettare" le eccezioni.
Questo esercizio richiedeva di sollevare un'eccezione, dunque un costrutto try/catch sarebbe completamente fuori luogo.

Per verificare che l'insieme non contenga Integer, è sufficiente controllare il risultato di uno dei due metodi maxInteger e minInteger. È inutile controllarli entrambi.

Visto che RuntimeException è non controllata, non è necessario dichiararla con throws nell'intestazione del metodo.

3. La classe `InsiemeNumeri` è implementata mediante un unico campo

```
private Number[] numeri
```

che si riferisce ad un array contenente i numeri presenti nell'insieme. Scrivete l'implementazione del metodo `maxInteger`. *Ricordate che grazie al meccanismo di unboxing è possibile applicare operatori di confronto come* `<` *e* `>` *a due riferimenti di tipo* `Integer` *(non* `Number`*!) per confrontare i valori degli oggetti associati. In alternativa, il confronto può essere effettuato utilizzando il metodo* `compareTo` *(il metodo è fornito da* `Integer`*, non da* `Number`*!).*

```
public Integer maxInteger() {
  Integer max = null;
  for (Number n: numeri)
    if (n instanceof Integer) {
      Integer i = (Integer) n;
      if (max == null || i > max)
        max = i;
    }
  return max;
}
```

Si esaminano uno dopo l'altro i riferimenti contenuti nell'array (ciclo for-each).
Per ogni riferimento a un oggetto di tipo Integer (verifica effettuata utilizzando l'operatore instanceof) si confronta il valore dell'oggetto associato con il massimo precedente (variabile max), aggiornando max nel caso l'oggetto considerato rappresenti un valore Integer maggiore.
Il confronto viene effettuato utilizzando l'operatore > che, come indicato nel testo dell'esercizio, è applicabile a operandi di tipo Integer, ma non di tipo Number. A tale scopo è necessaria la forzatura del riferimento n al tipo Integer, ottenendo il riferimento i utilizzato nel confronto.
La variabile max viene inizializzata a null. Tale valore verrà mantenuto fino a quando non viene individuato per la prima volta un oggetto di tipo Integer. In tal caso, eseguendo la selezione "if (max == null II ...", la condizione risulta vera (senza che sia effettuato il confronto i > max grazie al meccanismo della lazy evaluation) e a max viene assegnato il valore di tale Integer. Se non ci sono valori Integer, alla fine max conterrà il riferimento null, che verrà restituito come richiesto dal contratto del metodo.

ATTENZIONE:

È sbagliato inizializzare la variabile max assegnandole il primo elemento dell'array: non è detto che esso sia di tipo Integer!

È sbagliato iniziale la variabile max assegnandole zero: l'array potrebbe contenere anche (o solo) valori negativi

L'operatore di confronto > non è applicabile a Number, dunque è necessaria la forzatura (discorso analogo se si usa compareTo)

Il metodo statico parseInt della classe Integer riceve come argomento un riferimento String. Pertanto se si fornisce un Number come argomento il compilatore segnala un errore di tipo.

Si scrive "instanceof" e non "istanceof"

Negli esercizi seguenti supponete di disporre anche di una classe concreta di nome `Alfa`, sottoclasse di `Number`. `Alfa` possiede un *unico costruttore* che riceve come argomento un valore di tipo `int`. Tra i metodi di `Alfa` vi è `public int intValue()` che restituisce il valore specificato al momento della creazione dell'oggetto. Ad esempio, il metodo `intValue()` di un oggetto costruito invocando `new Alfa(123)` restituisce `123`.

4. Considerate le seguenti classi:

```
public class Beta extends Alfa {
  private int x, y;
  private static int z = 2;

  public Beta(int s, int t) {
    super(z);
    x = s;
    y = t;
    z = s + t - z;
  }

  public int intValue() {
    return super.intValue() + x + y;
  }

  public static int getStatico() {
    return z;
  }
}
```

```
class Prova {
  public static void main(String[] args) {
    System.out.println(Beta.getStatico());  //1
    Alfa a = new Beta(5, 8);
    System.out.println(a.intValue());       //2
    a = new Alfa(10);
    System.out.println(a.intValue());       //3
    System.out.println(Beta.getStatico());  //4
  }
}
```

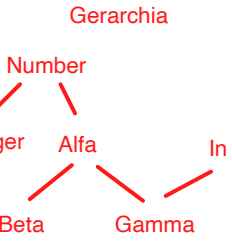
Scrivete in ogni riquadro l'output prodotto dall'istruzione di stampa seguita dal commento indicato:

//1	//2	//3	//4
2	15	10	11

Attenzione alla differenza tra campi e campi statici!

5. Oltre alle classi utilizzate negli esercizi precedenti, considerate una classe `Gamma` che estende `Alfa` e implementa un'interfaccia `In`.

a. Nel riquadro che precede ciascuna affermazione, scrivete V se l'affermazione è vera, F se è falsa:



F	Ogni istanza di `Beta` contiene esattamente 3 campi (oltre a quelli ereditati dalla superclasse)
V	`Number` è un supertipo di `Gamma`
F	`Gamma` deve ridefinire il metodo `intValue`
V	È possibile definire una sottoclasse astratta di `Number`
V	`Gamma` deve fornire l'implementazione dei metodi di `In`
V	`Alfa` deve fornire l'implementazione dei metodi astratti di `Number`
F	`Beta` deve fornire l'implementazione dei metodi astratti di `Number`
V	Ogni istanza di `Beta` contiene esattamente 2 campi (oltre a quelli ereditati dalla superclasse)
V	Se il codice sorgente della classe `Gamma` non contiene un costruttore allora il compilatore segnala un errore
F	`Gamma` può fornire l'implementazione dei metodi di `In`

I campi sono 2, x e y. Il campo z essendo statico appartiene all'intera classe.

Per implementare In, la classe Gamma DEVE fornire l'implementazione dei metodi dichiarati in In. Dire che Gamma "può" fornire l'implementazione significherebbe che Gamma ha la facoltà di farlo, ma anche di non farlo.

b. Considerate le seguenti dichiarazioni di variabile:

```
Integer w, Number n, Alfa a, Beta b, Gamma g, In i;
```

Nel riquadro accanto a ciascun assegnamento scrivete SI se l'assegnamento è compilato correttamente, NO se non è compilato correttamente (supponete che al posto di `...` vi siano gli argomenti opportuni):

NO	`g = n`
SI	`n = g`
SI	`n = w`
NO	`n = new Number(...)`
SI	`g = (Gamma) n`

Number è astratta!

SI	`a = (Gamma) i`
SI	`w = (Integer) n`
SI	`i = (Gamma) a`
NO	`a = (Gamma) b`
NO	`b = a`

c. Nel riquadro che precede ciascuna affermazione, scrivete V se l'affermazione è vera, F se è falsa:

F	Tutte le eccezioni di tipo `RuntimeException` sono controllate
F	I riferimenti agli oggetti sono sempre memorizzati nello heap
F	In Java è possibile definire nuovi tipi primitivi
F	I campi statici, come `z` della classe `Alfa`, sono memorizzati ~~nello heap~~
F	I parametri `s` e `t` del costruttore di `Beta` vengono memorizzati ~~nello heap~~
V	Durante l'esecuzione lo stack può contenere più record di attivazione di uno stesso metodo
V	Gli oggetti sono sempre memorizzati nello heap
V	Tutte le eccezioni di tipo `IOException` sono controllate
V	La variabile `a` del metodo `main` della classe `Prova` viene memorizzata nello stack
F	Il tipo `int[]` è primitivo
F	L'overloading dei metodi viene risolto in fase di esecuzione
F	Il linguaggio Java viene chiamato anche *bytecode*
F	Durante l'esecuzione lo stack contiene il codice dei costruttori e dei metodi
V	In Java è possibile definire nuovi tipi riferimento
F	In una stessa classe è possibile definire più metodi con la stessa segnatura, ma tipo restituito differente
V	In una stessa classe è possibile definire più metodi con la stesso nome, ma segnatura differente
V	In Java gli array sono oggetti
F	In Java un'interfaccia può possedere un costruttore
F	La Java Virtual Machine è un compilatore
V	La Java Virtual Machine è un interprete

Per definizione tutte le eccezioni di tipo RuntimeException (e sottoclassi) sono NON controllate

Nello heap vengono memorizzati gli oggetti. L'area in cui viene memorizzato un riferimento dipende da dove è stato dichiarato, ad esempio se il riferimento è una variabile locale di un metodo allora viene memorizzato, come tutte le variabili locali, nello stack.

nella memoria statica

nello stack (all'interno del record di attivazione del costruttore)

infatti IOException non è sottoclasse di RuntimeException, ma di Exception

all'interno del record di attivazione di main

È un tipo riferimento ad array

Viene risolto in fase di compilazione scegliendo la segnatura del metodo che dovrà essere eseguito. In esecuzione viene risolto l'overriding.

Il bytecode è un linguaggio simile a un assembler in cui il compilatore Java traduce i sorgenti. Viene poi interpretato dalla Java Virtual Machine

Lo stack contiene il codice ma i dati i organizzati in record di attivazione

Overloading: metodi con lo stesso nome, ma differente lista di argomenti (dunque segnatura differente). Non è possibile invece distinguere due metodi solo in base al tipo restituito.

La Java Virtual Machine (comando java) interpreta il bytecode prodotto dal compilatore Java

6. Considerate la dichiarazione di variabile `String[] nomi` e il seguente frammento di codice:

```
int x = 0;
try {
  x = nomi[x].length() / nomi[x].length();
} catch (ArithmeticException e) {
  x = x + 15;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 22;
} catch (NullPointerException e) {
  x = x + 29;
}
```

Ricordando che:
- `ArithmeticException` viene sollevata in caso di anomalie nel calcolo di operazioni aritmetiche,
- `ArrayIndexOutOfBoundsException` viene sollevata quando si tenti di accedere a una posizione inesistente in un array,
- `NullPointerException` viene sollevata quando si tenti di accedere a un oggetto tramite un riferimento `null`,
- `""` indica la stringa vuota,

indicate nel riquadro corrispondente, in ciascuno dei seguenti casi, il valore della variabile `x` dopo l'esecuzione:

(a) l'array riferito da `nomi` contiene (nell'ordine indicato) riferimenti a oggetti che rappresentano le stringhe `"formica"`, `"cane"`, `""`. — 1

(b) `nomi` contiene `null`. — 29

(c) l'array riferito da `nomi` contiene (nell'ordine indicato) riferimenti a oggetti che rappresentano le stringhe `""`, `"formica"`, `"cane"`. — 15

(d) l'array riferito da `nomi` è vuoto. — 22

7. Considerate il seguente metodo ricorsivo. Scrivete il risultato restituito dalle chiamate indicate nei due riquadri:

```
... int f(int x) {
  if (x <= 1)
    return 3;
  else
    return 3 * f(x / 2) + x;
}
```

f(2)	f(5)
11	38